

drop: un programma interattivo per generare frattali per accrescimento

<http://didawiki.di.unipi.it/doku.php/fisica/informatica/>

Progetto di recupero del corso di Informatica (CDL Fisica) 2016/17

Contents

1	Introduzione	1
1.1	Materiale in linea	1
1.2	Struttura del progetto e consegna . . .	1
1.3	Valutazione del progetto	1
2	Drop: una semplice forma frattale per accrescimento	2
2.1	Le quattro adiacenze	2
2.2	Generazione della forma	2
2.3	Esempi	3
2.3.1	Esempi senza ostacoli	3
2.3.2	Esempi con ostacoli	3
3	Il progetto	4
3.1	Le opzioni di drop	4
4	Come procedere	5
5	Codice e documentazione	5
5.1	Vincoli sul codice	5
5.2	Formato del codice	5
5.3	Relazione	5

Eventuali chiarimenti possono essere richiesti consultando i docenti l'orario di ricevimento e per posta elettronica.

1.2 Struttura del progetto e consegna

La consegna del progetto avviene *esclusivamente* per posta elettronica, attraverso il target **consegna** del **Makefile** contenuto nel kit di sviluppo del progetto (scaricabile dalla pagina degli assegnamenti) e deve contenere: tutti i file sorgente (**myrand.c**, **drop.c**, **file.c** e **main.c**), la relazione in formato PDF (file **relazione.pdf**) e un file **gruppo.txt** che specifica numero di matricola, cognome, nome ed indirizzo di mail degli autori (max 2), secondo il formato (notare la virgola come separatore)

Zini, Gino, 123456, zini@madoc.it
Bixio, Nino, 456378, bixio@gmail.com

secondo le indicazioni del **README** nel kit di progetto. Tutti i file del progetto si devono trovare nella directory **DROP/** e non sono ammesse sottodirectory.

*I progetti che non rispettano il formato o non consegnati con il target **consegna** non verranno accettati.*

1 Introduzione

Il progetto consiste nello sviluppo di un programma C che genera e memorizza frattali per accrescimento.

1.1 Materiale in linea

Tutto il materiale si trova alla pagina web del corso.

1.3 Valutazione del progetto

Al progetto viene assegnato un punteggio da 0 a 30 in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice

- efficienza e robustezza del software
- aderenza alle specifiche
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione

La discussione del progetto in sede di orale tenderà a stabilire se lo studente è realmente l'autore e verterà su tutto il programma del corso. Il voto dell'orale (ancora da 0 a 30) fa media con la valutazione del progetto per delineare il voto finale.

2 Drop: una semplice forma frattale per accrescimento

Drop è una semplice forma frattale basata su un modello di caduta e sedimentazione di particelle in un'area bidimensionale in cui sono presenti alcuni ostacoli rettangolari.

Utilizziamo una matrice $N \times M$ che rappresenta una griglia di possibili sottoaree (*celle* da ora in poi) in cui si trova la particella durante la caduta. Gli elementi della matrice possono assumere tre valori: **EMPTY** (se non contengono ancora nessuna particella sedimentata), **FULL** (se contengono già una particella) o **OBSTACLE** (se contengono parte di un ostacolo rettangolare).

2.1 Le quattro adiacenze

Per ogni cella (i, j) della matrice sono definite quattro tipi di adiacenza

NONE *nessuna adiacenza* in questo caso l'insieme delle celle adiacenti è vuoto (questa adiacenza serve per simulare la semplice sedimentazione delle particelle sul fondo)

CROSS *adiacenza a croce* in questo caso le celle adiacenti sono (parentesi tonde)

$$\begin{array}{ccc} & (i-1, j) & \\ (i, j-1) & <i, j> & (i, j+1) \\ & (i+1, j) & \end{array}$$

DIAGONAL *adiacenza diagonale* in questo caso le celle adiacenti sono (parentesi tonde)

$$\begin{array}{ccc} (i-1, j-1) & & (i-1, j+1) \\ & <i, j> & \\ (i+1, j-1) & & (i+1, j+1) \end{array}$$

BOTH *adiacenza a croce & diagonale* in questo caso le celle adiacenti sono quelle ottenute usando sia **CROSS** che **DIAGONAL**.

2.2 Generazione della forma

La generazione della forma frattale procede come segue.

All'inizio tutta l'area di caduta è vuota (a parte le aree in cui si trovano gli ostacoli). Viene fissata una adiacenza fra le quattro descritte in Sec. 2.1, che sarà usata per tutta la simulazione.

La generazione avviene simulando la caduta di una serie di particelle sempre partendo dalla cella P_0 di coordinate $(0, \lfloor M/2 \rfloor)$ e spostandosi ad ogni passo nella riga successiva della griglia in modo da scegliere casualmente fra le tre celle successive libere. Più precisamente, ad ogni passo, se la particella si trova nella cella $P_i = (i, j)$ sia U l'insieme delle celle **EMPTY** fra la tre celle candidate

$$(i+1, j-1) \quad (i+1, j) \quad (i+1, j+1)$$

la prossima posizione della particella è scelta in modo equiprobabile fra le celle in U . Ad esempio nel caso in cui le tre celle candidate contengano rispettivamente

EMPTY EMPTY FULL

allora l'insieme U è dato da $U = \{(i+1, j-1), (i+1, j)\}$ e scegliamo fra i due elementi di U con probabilità $\frac{1}{2}$.

La caduta della particella si arresta quando si verifica uno dei seguenti casi:

1. la particella si trova all'ultima riga (la $N-1$) (sedimenta sul fondo)
2. l'insieme U è vuoto
3. esiste almeno una cella **FULL** o **OBSTACLE** adiacente a (i, j) considerando l'adiacenza scelta all'inizio della simulazione.

In tutti questi casi, la cella (i, j) diventa a sua volta piena (FULL).

La generazione termina o dopo aver fatto cadere un numero fissato di particelle (nel seguito la caduta di una singola particella viene chiamata *step*), oppure quando la cella iniziale P_0 contiene già una particella sedimentata.

2.3 Esempi

2.3.1 Esempi senza ostacoli

A esempio, una possibile evoluzione con 5 righe e 5 colonne, senza ostacoli e con adiacenza **CROSS** è la seguente (la stessa del primo assegnamento). Cella iniziale della prima particella (per leggibilità abbiamo indicato con un punto ('.') le celle vuote e con asterisco ('*') le celle piene:

```
..*..
....
....
....
....
```

Cella in cui si sedimenta la prima particella

```
.....
.....
.....
.....
...*.
```

Situazione dopo due cadute:

```
.....
.....
.....
.....
...**.
```

Situazione dopo 8 cadute

```
.....
..*..
..**.
..**.
*..**.
```

Da notare che se la particella raggiunge una delle celle appartenenti alla prima o all'ultima colonna della matrice ha solo due (2) celle in cui può spostarsi al passo successivo.

Di seguito un secondo esempio ottenuto con adiacenza **BOTH**

```
.....*.....
.....*.....
.....*.....
.....*.....
.....*.....
.....*.*.....
.....*.*.....
.....*.***.....
.....*.***.....
.....*.***.....
.....*.***.....
.....*.***.....
.....*.***.....
.....*.***.....
```

e infine uno con adiacenza **NONE**

```
.....
.....
.....
.....*.....
.....****.....
.....*****.....
.....*****.*.....
```

2.3.2 Esempi con ostacoli

Supponiamo di avere tre ostacoli (nel seguito segnalati da una chiocciola @), una possibile evoluzione con adiacenza **NONE** è la seguente

```
.....
.....@@@.....
.....@@@.....
.....@@@.*.....
.....@@@.@@@.*.....
.....@@@.....@@@@@.....
.....@@@.....**@@@@@.....
.....@@@.....***@@@@@@@@@@@@@@@@
.....*.*****@@@@@@@@@@@@@@@@
```

mentre con l'adiacenza **DIAGONAL** otteniamo

```
.....
.....@@@.....
.....@@@.....
.....@@@.....
.....@@@.@@@.*.....
.....@@@.....**.....@@@@@.....
.....@@@.....***.*.@@@@@.....
.....@@@.*.....**.*@@@@@@@@@@@@@@@@
.....**.....*.***@@@@@@@@@@@@@@@@
```

3 Il progetto

Lo scopo del progetto è lo sviluppo un programma per la generazione di forme frattali secondo il modello Drop. L'eseguibile corrispondente si chiamerà **drop**.

Ogni esecuzione di **drop** effettua le seguenti operazioni:

- legge da linea di comando un insieme di opzioni che fissano le caratteristiche della generazione (specificate più avanti)
- genera la forma frattale
- la stampa sullo standard output
- (se richiesto) salva la matrice generata su file
- termina l'esecuzione

3.1 Le opzioni di drop

drop effettua la generazione di una forma frattale e stampa il frattale ottenuto sullo standard output, può essere attivato con le opzioni mostrate in Fig. 1, dove fra parentesi quadre sono inserite le parti opzionali. Il significato è il seguente:

-n iter: deve essere sempre presente e stabilisce il numero di step che devono essere eseguiti durante la generazione (numero di particelle da far cadere) a partire dallo stato iniziale dell'area di caduta che può essere letto da file (**-f**) oppure fornito con le opzioni **-n** e **-m**. Se nessuna di queste opzioni è presente si simula un'area vuota 20×20 .

-a ad: l'opzione **-a** (se presente) permette di scegliere l'adiacenza con cui effettuare la generazione. **ad** può avere come valori **NONE**, **DIAGONAL**, **CROSS** o **BOTH**. Se l'opzione non è presente sulla linea di comando si utilizza sempre l'adiacenza **NONE**.

-f infile: l'opzione **-f** (se presente) permette di caricare da file una matrice che rappresenta l'area di caduta, comprensiva di ostacoli e di eventuali particelle già sedimentate. La matrice si trova nel file **infile**, dove viene rappresentata in forma compatta con un formato deciso dallo studente (il formato usato deve essere specificato nella relazione). Questa opzione è alternativa alle opzioni **-n** e **-m**.

-n N -m M: le opzioni **-n** e **-m** permettono di costruire l'area di caduta vuota fornendo il numero di righe **N** e il numero di colonne **M** della matrice che la rappresenta. Queste opzioni devono essere entrambe presenti e sono incompatibili con l'opzione **-f**

-u file_ostacoli: l'opzione **-u** permette di aggiungere gli ostacoli nel file **file_ostacoli** all'area di caduta, il formato degli ostacoli nel file è lo stesso utilizzato nel file di input di **test.six.c**, ovvero quattro interi unsigned che rappresentano le coordinate dell'ostacolo un alto a sinistra e in basso a destra,

-o outfile: l'opzione **-o** salva su file lo stato della matrice alla fine della generazione secondo il formato compatto scelto dallo studente (lo stesso dell'opzione **-f**).

Ad esempio, le seguenti sono invocazioni corrette di **drop**:

drop 250

effettua 250 step di generazione su una matrice 20×20 senza ostacoli e con adiacenza **NONE**.

drop 250 -u ciccio -n 40 -m 21

effettua 250 step di generazione su una matrice 40×21 con gli ostacoli specificati nel file **ciccio** e con adiacenza **NONE**.

drop 250 -u ciccio -a BOTH

effettua 250 step di generazione su una matrice 20×20 con gli ostacoli specificati nel file **ciccio** e con adiacenza **BOTH**.

drop -u ciccio -a DIAGONAL -f savefile 250

effettua 250 step di generazione su una matrice letta dal file **savefile** a cui sono stati aggiunti gli ostacoli specificati nel file **ciccio** e con adiacenza **DIAGONAL**.

drop 21 -o outfile -u ostacoli -f savefile

effettua 21 step di generazione su una matrice letta dal file **savefile** a cui sono stati aggiunti gli ostacoli specificati nel file **ostacoli**, con adiacenza **NONE** la matrice risultante è salvata nel file **outfile**.

Le opzioni possono comparire in un ordine qualsiasi. Si suggerisce di utilizzare la funzione di libreria **getopt** utilizzata per la l'analisi delle opzioni dei comandi della shell.

In caso di opzioni errate deve essere segnalato l'errore e stampato un opportuno messaggio di uso.

Facoltativamente, lo studente può aggiungere altre opzioni (ad esempio **--help**) che devono essere documentate nella relazione finale.

```
./drop n_iter [ -a ad ] [ -f infile ] [ -o outfile ] [ -u file_ostacoli ] [ -n N -m M ]
```

Figure 1: Argomento e opzioni linea di comando di **drop**.

4 Come procedere

Lo studente deve prima realizzare (nel file **myrand.c**) le funzioni i cui prototipi si trovano nel file **myrand.h** testandole con i test forniti dai docenti (test1). Poi deve sviluppare (nel file **drop.c**) le funzioni i cui prototipi si trovano nel file **drop.h** testandole con i test forniti dai docenti (test2-test5). Successivamente deve sviluppare (in **file.c**) le funzioni i cui prototipi si trovano nel file **file.h** testandole con i test forniti dai docenti (test6).

Infine (in **main.c**) deve sviluppare il programma **drop** con le opzioni specificate nella Sezione 3.1, utilizzando le funzioni già sviluppate per portare a termine le varie operazioni richieste dalle varie combinazioni di opzioni.

Ricordiamo anche di leggere il file **README** nella directory **DROP** che contiene le indicazioni precise su come sviluppare ed eseguire il progetto.

5 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

5.1 Vincoli sul codice

Il codice consegnato deve rispondere ai seguenti vincoli:

- il codice deve utilizzare le strutture dati definita nel file ***.h** contenuto nel kit;
- il codice delle funzioni sviluppate nei file ***.c** deve superare tutti i test previsti nel **Makefile** contenuto nel kit;
- il codice deve compilare senza errori o warning gravi utilizzando le opzioni **-Wall -pedantic**
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune **#define** o **enum**)

5.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- un'intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore;
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore, valori restituiti
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali non di uso ovvio, che spieghi a cosa servono

5.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *NON deve ripetere le presenti specifiche ma deve rendere comprensibile il lavoro svolto per realizzarle ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli*. In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali) e loro motivazioni
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- il formato usato nel salvataggio su file
- eventuali parti opzionali inserite
- le difficoltà incontrate e le soluzioni adottate
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- istruzioni per l'utente su come compilare/eseguire ed utilizzare il codice (in quale directory deve essere attivato, quali sono le assunzioni fatte etc)

La relazione deve essere in formato PDF e il file si deve chiamare **relazione.pdf** tutto minuscolo.